$2.95 in USA

UNDERSTANDING ATARI® GRAPHICS

Including GTIA Graphics

Learn to Create Beautiful and Practical Graphics
With These Easy to Follow Instructions
For Models 400 and 800

An Alfred Handy Guide

050N 143
PRICE 2.95
A 4D 6368

By Michael Boom

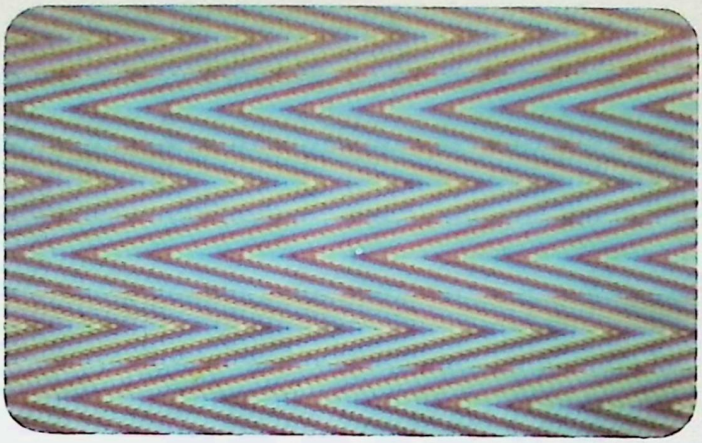


PLATE 1 Rotating colors in this mode-10 display of Program 7 gives the illusion of motion.

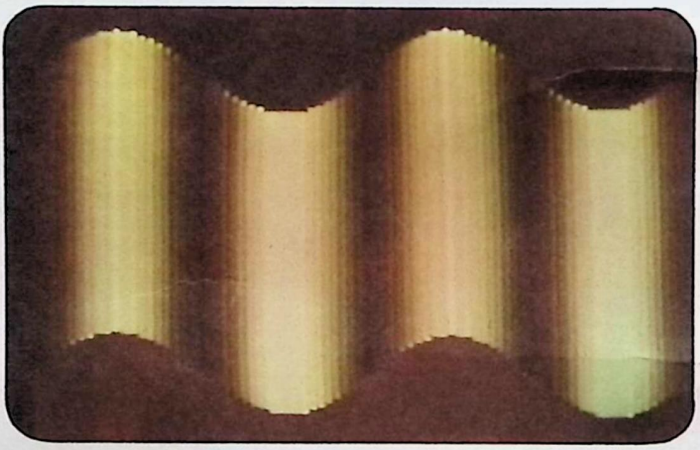


PLATE 2 Delicate shading in this mode-9 display of Program 8 creates a three-dimensional image.

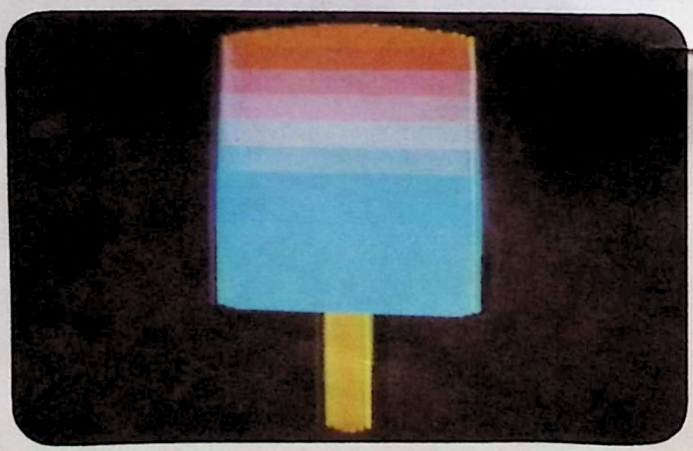


PLATE 3 An entire spectrum of colors can be displayed simultaneously in the mode-11 Program 9.

UNDERSTANDING ATARI GRAPHICS

by Michael Boom

AN ALFRED HANDY GUIDE

Computer Series Editor:
George Ledin Jr.

List of Programs 3
List of Plates and Figures 4
1. Introduction 5
2. Basic Concepts and Terms 7
Atari Display Modes 7
Pixels 8
Screen Address 9
Pixel Loading 10
Character Sets 11
Color Registers 11
The Cursor 13
3. Graphics Commands 14
Formatting Commands 14
Graphics 14
Setcolor 17
Color 17
Input/Output 18
Plot 18
Drawto 18
Position 19
Print 19
Using Print in Text Modes 20
Using Print in Graphics Modes 22
Locate 22
Put/Get 23
XIO 25
Command Summary 27
4. The GTIA Chip and Graphics Modes 9, 10, and 11 28
5. Tips and Tricks 34
Appendix A: Display Mode Information 36
Text Modes 36
Graphics Modes 38
Appendix B: ATASCII Code Chart 41

ALFRED PUBLISHING CO., INC.
SHERMAN OAKS, CA 91403

This Handy Guide is not a publication of Atari and should not be used in lieu of the instruction manuals that accompany their products. All information regarding Atari graphics may not be accurate or completely up to date.

Editorial Supervision: Joseph Cellini

Cover Design: Paula Bingham Goldstein

Production Management: Michael Bass & Associates

Copyright © 1982 by Alfred Publishing Co.,Inc.
Printed in the United States of America.
All rights reserved. No part of this book shall be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information or retrieval system without written permission of the publisher.

Alfred Publishing Co.,Inc.
P.O. Box 5964
15335 Morrision Street
Sherman Oaks, CA 91413

Library of Congress Cataloging in Publication Data

Boom, Michael.
Understanding Atari graphics.

(An Alfred handy guide)
1. Atari computer—Programming. 2. Computer graphics.
I. Title.
QA76.8.A82B66 1982 001.64'43 82-18463
ISBN 0-88284-224-2

LIST OF PROGRAMS

Program 1	Chapter 2, p. 8
Program 2	Chapter 3, p. 16
Program 3	Chapter 3, p. 18
Program 4	Chapter 3, p. 24
Program 5	Chapter 3, p. 26
Program 6	Chapter 4, p. 28
Program 7	Chapter 4, p. 31
Program 8	Chapter 4, p. 32
Program 9	Chapter 4, p. 33

LIST OF PLATES AND FIGURES

Plate 1. Mode-10 display of Program 7. Inside front cover

Plate 2. Mode-9 display of Program 8. Inside front cover

Plate 3. Mode-11 display of Program 9. Inside front cover

Plate 4. Chart of available hues using Atari SETCOLOR command. Inside back cover

Figure 1. Atari 400 computer system. Chapter 1, p. 5

Figure 2. Atari 800 computer system. Chapter 1, p. 5

Figure 3. Different forms of computer output. Chapter 2, p. 7

Figure 4. Animal drawn with pixels. Chapter 2, p. 7

Figures 5 A & B. Low and high resolution in modes 3 and 7. Chapter 2, p. 9

Figure 6. Pixel at coordinates 12,4. Chapter 2, p. 10

Figure 7. Character set for READY prompt. Chapter 2, p. 11

Figure 8. Color register default colors. Chapter 2, p. 12

Figure 9. Text window and graphics window. Chapter 3, p. 15

Figure 10. Color register chart. Chapter 3, p. 18

Figures 11 A & B. Program 4 in mode 3 and mode 0. Chapter 3, p. 25

Figure 12. GTIA and mode 8 pixels. Chapter 4, p. 28

Figure 13. Chart of mode 10 pixel value-color register equivalencies. Chapter 4, p. 31

1. INTRODUCTION

The Atari personal computer is perhaps the most powerful video graphics computer available today for under $2000. Its designers have given it capabilities far beyond those of other personal computers in the same price bracket. It can outperform many computers that are much more expensive.

These graphics capabilities offer the opportunity to use good video displays with your programs. Good graphics can dress up ordinary business programs,

FIGURE 1. Atari 400 computer system.

FIGURE 2. Atari 800 computer system.

make educational programs more enticing, and make or break a game program. A few graphics-display additions will communicate ideas to a program user much more quickly and conveniently than a strictly textual presentation will.

This Handy Guide has been written to teach beginning and intermediate programmers how to use Atari BASIC graphics. You will need to know Atari BASIC, the language most often used on the Atari. It will also be convenient to have the Atari BASIC Reference Manual at hand as a reference, although this is not absolutely necessary. To learn from this Handy Guide effectively, you should have access to an Atari 400 or Atari 800 computer. While it is possible to learn the graphics commands without actually trying them on a computer, entering the sample programs and seeing the results on the TV screen makes it easy to understand just how the commands work.

The material following this introduction is arranged to take you through Atari BASIC graphics programming in a logical and thorough manner. We first discuss basic terms and concepts important to graphics programming. Then we examine each of the Atari BASIC commands used for generating video graphics:

- GRAPHICS
- SETCOLOR
- COLOR
- PRINT
- PLOT
- DRAWTO
- POSITION
- LOCATE
- PUT
- GET
- XIO

Following this is a section on new graphics capabilities available on Atari computers equipped with a GTIA chip. The final section presents some ways to make graphics programming easier and more effective. In the appendices, there are charts for the display modes and the ATASCII code. Plate 4 on the inside back cover is a color table showing available hues using the Atari SETCOLOR command.

2. BASIC CONCEPTS AND TERMS

Your computer usually presents information to you through your television screen. It can also pass information through a printer, the sounds from your television speaker, or even an attached voice synthesizer (see Figure 3). The main advantages of video presentation are that it is very fast, relatively inexpensive, and presents large amounts of information at one time.

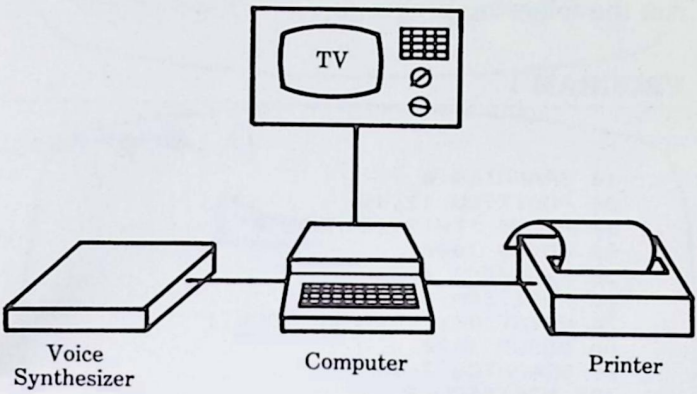


FIGURE 3. Different forms of computer information systems output.

ATARI DISPLAY MODES

With the Atari, video information can be presented in many different ways. You are probably most familiar with a textual presentation, in which the computer prints words in light blue on a dark blue background. Some of the programs or games you own may present information graphically, with multicolored pictures or charts. The Atari has different methods of video presentation, each of which is known as a *display mode.* In the Atari BASIC language there are 9 modes available, numbered 0

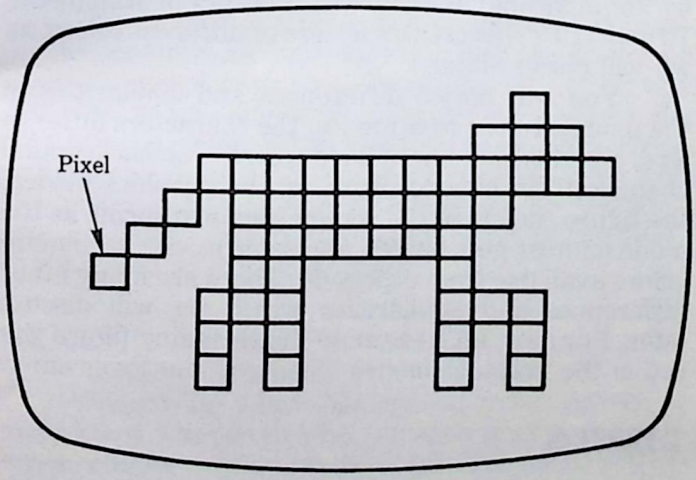


FIGURE 4. Animal drawn with pixels.

through 8. (Newer Atari computers may have 12 modes available, numbered 0 through 11. Modes 9 through 11 will be covered in Chapter 4 on GTIA modes.) Three of these modes are *text modes* (0 to 2), which present information in *characters*—e.g., numbers, letters, punctuation, symbols. (Mode 0 is the normal (or default) mode usually seen on your display.) The other six modes are *graphics modes*, which present information in colored blocks known as *pixels* (short for "picture elements"), as shown in Figure 4.

To see the display modes available, enter and run the following program:

PROGRAM 1

```
10 GRAPHICS 0
20 POSITION 12,12
30 PRINT "THIS IS MODE 0"
40 GOSUB 1000
50 GRAPHICS 1+16
60 POSITION 3,12
70 PRINT #6;"this is MODE 1"
80 GOSUB 1000
90 GRAPHICS 2+16
100 POSITION 3,6
110 PRINT #6;"this is MODE 2"
120 GOSUB 1000
200 FOR K = 3 TO 8
210 GRAPHICS K
220 COLOR 1
230 PLOT 10,5: DRAWTO 10,15: DRAWTO
30,15: DRAWTO 10,5
240 PRINT "THIS IS MODE ";K
250 GOSUB 2000
260 NEXT K
270 GRAPHICS 0
280 END
1000 FOR I = 1 TO 1000: NEXT I: RETURN
2000 GOSUB 3000
2010 IF INT (K/2)=K/2 THEN 2030
2020 SETCOLOR 0,12,10: GOSUB 3000
2030 SETCOLOR 0,9,4: GOSUB 3000
2040 SETCOLOR 0,0,0: GOSUB 3000
2050 RETURN
3000 FOR I = 1 TO 500: NEXT I: RETURN
```

Be sure to use inverse letters (indicated here by an underline) and lowercase letters in statements 70 and 110. (This creates letters of different colors, as we will discuss later.)

You will notice differences and similarities in the modes. In the text modes, the characters differ in size, and modes 1 and 2 have colors. The background changes from blue to black. In the graphics modes, the figure shown on the screen seems to shrink as the mode number gets larger, and some modes have more colors available than others do. There are many other differences and similarities which we will discuss later. For now, let's examine the shrinking figure you see in the graphics modes when you run Program 1.

PIXELS

The figure drawn on the screen is a simple triangle, and the pixels that make it up can be seen clearly (Figure 5A). As the mode number increases

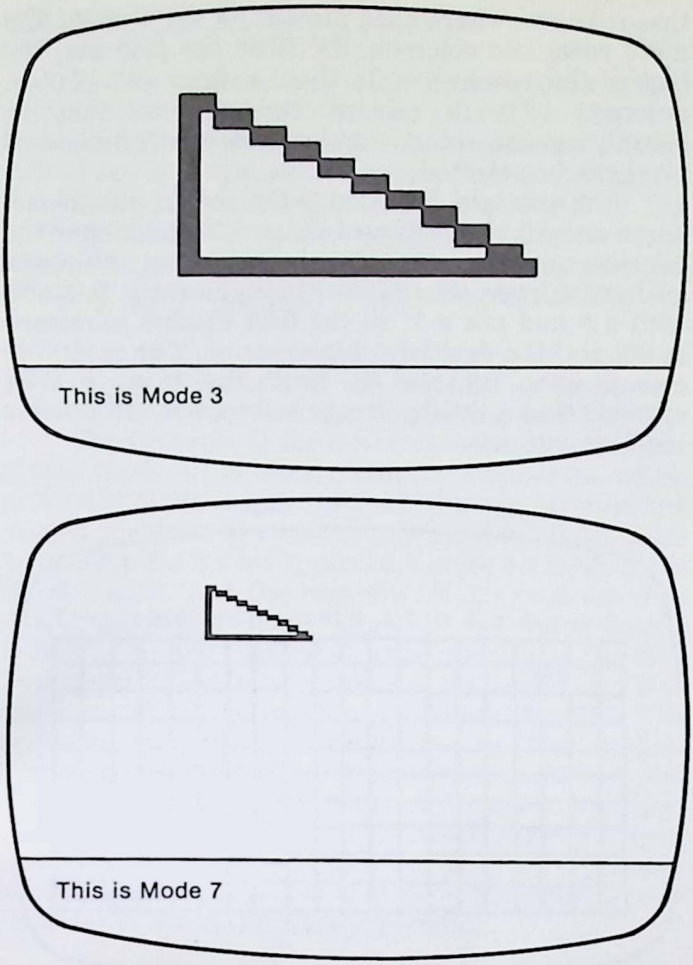


FIGURE 5 A & B. Low and high resolution in modes 3 and 7.

the figure gets smaller, because the size of the individual pixels shrinks (Figure 5B). The jagged line begins to look smoother. This demonstrates an important concept—*resolution,* or how accurately and finely a picture can be drawn. The smaller the pixels are, the finer the picture becomes; the larger the pixels, the coarser the picture. Modes using large pixels are referred to as *low-resolution modes.* Modes using small pixels are called *high-resolution modes.*

The concept of a pixel as a rectangular element on the screen is an important one and should not be limited to graphics modes. Therefore, let's think of a *text pixel* as an element in a text mode that can display one character. When you turn your computer on and it displays READY, you see five pixels filled with the letters R,E,A,D, and Y. Pixels can be "empty," too: an empty pixel is the same color as the background, and so cannot be seen.

SCREEN ADDRESS

When your Atari displays on a TV screen, it must know what to display for each pixel—a character, a color, or a space (invisible pixel). How does it do this? Each mode, whether text or graphic, sets aside a place in its memory to keep track of each pixel. It divides the screen into a grid of columns and rows, so

that it knows where each pixel is on the screen. The more rows and columns, the finer the grid and the higher the resolution. It then assigns consecutive numbers to each column and row so that by identifying one column and one row, an individual pixel can be selected.

For example, let's divide the screen into 20 columns and 10 rows (Figure 6), and then number the columns and rows. The Atari starts from the upper lefthand corner, as you do when you read. It starts with a 0 and not a 1, so the first column is column no. 0, and the first row is row no. 0. The twentieth column is no. 19, and the tenth row is no. 9. This system takes a little getting used to, but will become familiar with use.

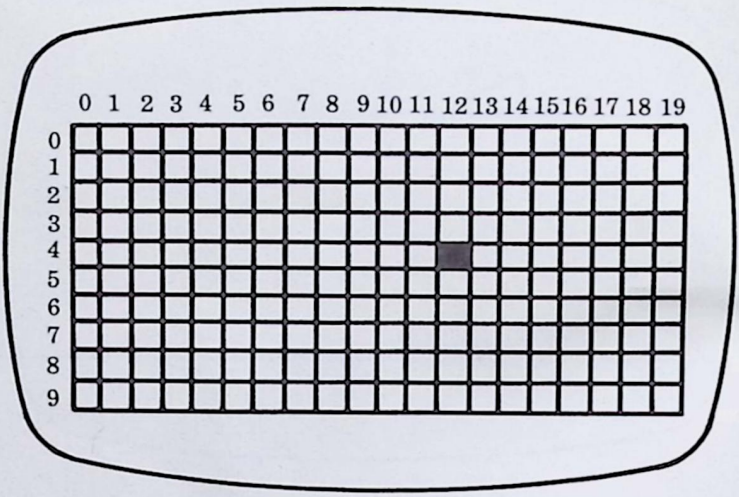


FIGURE 6. Pixel at coordinates 12,4.

Each of the squares in the grid is a pixel. Each pixel has a screen address, which is its column number followed by its row number. These numbers, called *coordinates,* are symbolized by *"x,y,"* where x is the column number and y is the row number. They are always separated by a comma. It is important to remember that the column number always comes first.

To find a pixel on the grid, take the coordinates and find where column x intersects row y. In Figure 6, the pixel marked is at coordinates 12,4: the thirteenth column over, the fifth row down.

Now the computer has a space in its memory for each pixel coordinate from 0,0 to 19,9 (200 pixels in all). How does it know what to display for each pixel? Very simple: it uses a numerical code.

PIXEL LOADING

At each pixel address in the Atari memory is a number that tells the Atari what to put into that particular pixel on the screen. In the text modes, each number refers to a character; in the graphics modes, each number refers to a color. How does the Atari know which character or color is assigned to a number? To explain, we will look at two memory tools used in the Atari—the *character set* and the *color register.*

CHARACTER SETS

In the text modes, each figure (character) is referred to by a number from 0 to 255. The code associating the numbers with the characters is called ASCII (an acronym for American Standard Code for Information Interchange). Atari has added to this code so that the special graphics characters and inverse letters on the computer can be represented. For the entire Atari ASCII code (ATASCII), see Appendix B on p. 42.

For an example of ATASCII at work, see how your computer displays the READY prompt that appears when you turn it on, referring to Figure 7.

The first row at the top of the screen is all empty pixels. Each one is loaded with the number 32, which is ATASCII for a space. READY is displayed in the second row (row 1, remember?). Pixel 0,1 is 32; pixel 1,1 is 32; pixel 2,1 is 82; pixel 3,1 is 69; 4,1 is 65; 5,1 is 68; 6,1 is 89; and the rest are all 32s (spaces). The Atari interprets these numbers as space-space-R-E-A-D-Y and displays each character in the appropriate pixel so that you can read READY. It is not normally necessary for you to know the ATASCII codes to generate characters in the text modes. Simply using certain keys or combinations of keys will work. The Atari will automatically convert your keystrokes to the appropriate ATASCII code numbers and load those numbers into the pixels. For certain applications, however, it will be helpful for you to know what these numbers represent.

COLUMN:	0	1	2	3	4	5	6	7	8	9
ROW: 0	32	32	32	32	32	32	32	32	32	32
1	32	32	82	69	65	68	89	32	32	32
2			R	E	A	D	Y			

These Values
Create this Display:

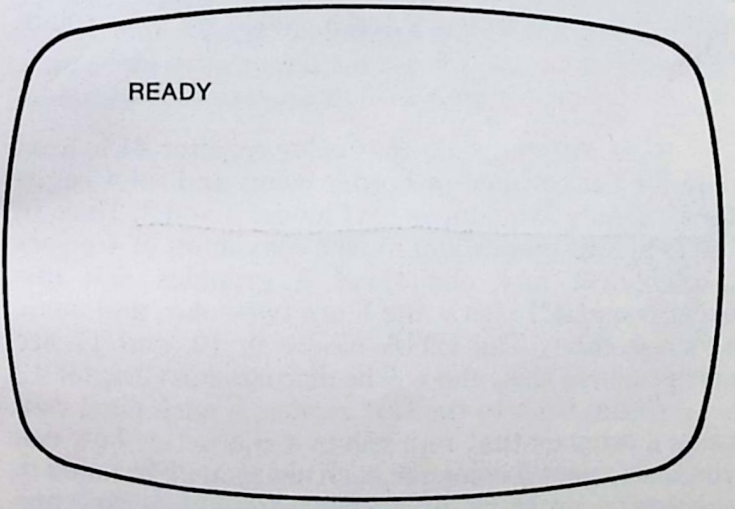


FIGURE 7. Character set for READY prompt.

COLOR REGISTERS

In the graphics modes, the Atari interprets each number in a pixel as a color, not a character. In Atari

BASIC, a color is defined by combining a hue and a luminance. You can see the 16 hues available in Plate 4 (back cover). The luminance is the shade or brightness of the hue you would like. Since there are 16 different hues available in 8 different luminances each, we have 128 different colors to work with.

The Atari does not put numbers from 0 to 127 into pixels and then interpret them as 128 separate colors. Instead, Atari BASIC uses only 5 colors at a time (or 9 colors in GTIA mode 10). This saves much memory, and allows for much easier use of color graphics. This is how the system works: We can choose the colors we wish to work with, and then store them in color registers numbered from 0 to 4. Each color register is like a pot of paint. As the computer reads numbers from 0 to 4 stored in pixels, it refers to the color register specified by each number and then "paints" the pixel with the color contained in that color register. For example, let's say color register #0 is assigned orange. All pixels that are loaded with this color register will appear orange. If color register #0 is then changed to blue, then all of those pixels will change to blue. This happens virtually instantaneously, since the computer "refreshes" the screen 60 times a second. Color registers make it very easy to change the color of many pixels at once.

The color registers can be assigned specific colors using the SETCOLOR command, which we will discuss in Chapter 3. Otherwise, the color register default colors (those which are assigned automatically by the computer everytime it is turned on) are as follows:

Color Register #	Default Color
0	orange
1	green
2	blue
3	red
4	black

FIGURE 8. Color register default colors. (Exceptions to this are modes 0 and 8, which we will explain later.)

It is worth noting that color register #4 is used only for background or border color, and color register #3 is only available in text modes 1 and 2. Thus, in the graphics modes you have a maximum of 4 colors available at any one time: 3 graphics and one background. (Modes 4 and 6 are two-color, and mode 8 is one-color. The GTIA modes 9, 10, and 11 are exceptions to this, and will be discussed in Chapter 4.)

Going back to the text modes, if each pixel contains a number that represents a character, how can you also select a color for each character? In mode 0, there is no problem: all 256 characters have the same color register. In modes 1 and 2, characters can be displayed in one of four colors. Therefore, in these modes the Atari limits us to 64 different characters (capital letters, numerals, punctuation marks, and other symbols such as dollar signs and asterisks) and assigns a different color register to each group. Each of these groups is then assigned 64 numbers from a

range of 0 to 255, replacing the regular ATASCII character assignments. Now when the Atari interprets a number from 0 to 255 in a pixel, it interprets it as one of the 64 characters in one of 4 colors set by the color registers 0 to 3. For example, with the default colors for the color registers, a 65 loaded into a pixel will yield an orange "A," a 97 will give a green "A," 193 will be blue, and 225 will be red. For a summary of these characters, their colors, and code assignments, refer to Appendices A and B.

Try GR 1
GR 2

THE CURSOR

Now we know how the Atari keeps track of individual pixels and screen display, but how do we go about changing the display so that it will show the text or graphics we want? We simply change the numbers in various pixels so that the appropriate character or color is shown. The most convenient way to do this is through a display device called a *cursor*—the small white rectangle that shows where you are typing on the screen in mode 0. You may find it convenient to think of the cursor as a door to any pixel on which it happens to be resting. We can merely open the pixel and look at the number inside, or we can reach inside and change the number to the one we want. When we type in characters at the keyboard, what we are actually doing is sending a number from the keyboard to the cursor, which inserts that number into the pixel and then moves on to the next one.

In mode 0 we can see the cursor as the inverse of the character it is resting on. In the other modes the cursor is invisible, but it is still there, moving from pixel to pixel at our command, looking at and changing their contents. In the next section we will learn how to control the position and function of the cursor.

We can look at the display screen, then, as a grid of pixels. Each pixel is filled with a number, which the Atari, after referring to a color register or a character set, interprets as a color or a character. We can change the display by changing the numerical contents of the pixels, and we use the cursor as a means of changing or looking at individual pixels.

3. GRAPHICS COMMANDS

To aid in our examination of the graphics commands used in Atari BASIC, we will divide these commands into two types: *formatting commands,* which prepare display conditions for later commands, and *input/output commands,* which actually change the data in the display pixels.

Formatting commands are not used for changing pixel data. They prepare display conditions for input/output commands, acting in a manner similar to an artist preparing to paint a picture. What medium will the painter employ: oil, watercolor, pencil, ink? Should the picture be put down on canvas, rough paper, silk, or some other background? What colors, if any, should be used? These questions must be answered before the painter can begin painting. In a similar manner, when we wish to select our display conditions, or *format,* we use formatting commands to tell the computer what colors, what pixel size, and what mode to use.

Input/output commands correspond to the brush strokes (or pen or pencil strokes) of the artist. Which of the colors available should be used? Shall I paint a line? In what direction will the line go? Should this area be filled in? We command the computer to perform similar functions using input/output commands —so named because they take numbers in and out of pixels to change the way they display.

FORMATTING COMMANDS

GRAPHICS

We use this command to select a display mode. Its format is GRAPHICS *m*, where *m* is the mode number we wish to select. In an earlier program, we looked at the nine different modes available. Now we will assign the numbers used in Atari BASIC and look at their general characteristics.

MODE 0

Mode 0 is the default mode. When we first turn on the computer, it is automatically in mode 0. This mode is a text mode, and can print 24 lines of 40 characters each. Its default colors are blue blackground with white letters (actually, a very light blue).

MODE 1

This is also a text mode. If you enter the command GRAPHICS 1 you will see that this mode has a split screen. There is a small strip of blue at the bottom, which prints characters in the same way that mode 0 does (see Figure 9). This is called a *text window* and is put there so that you can conveniently communicate with the computer using the keyboard. The rest of the screen is used for display. This mode-0 text-window is present in all modes from 1 to 8.

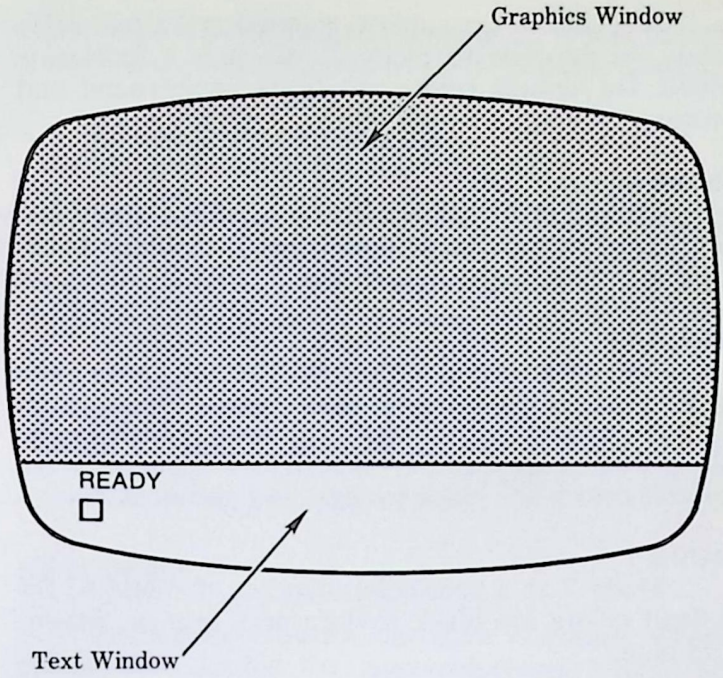


FIGURE 9. Text window and graphics window.

The black part of the screen is the *graphics window*. (To get back to mode 0, hit the SYSTEM RESET button or enter GRAPHICS 0.) This is where we can print text in mode 1. The characters here are the same height as mode 0 but are twice as wide. The graphics window can print 20 lines of 20 characters each. (It would be 24, but the text window takes up 4 lines.) It can print the characters in 4 different colors. Its default colors are orange, green, blue, and red characters, with a black background. (All colors will vary somewhat, depending on the TV set used.)

MODE 2

This is a text mode. It displays characters that are twice as wide *and* twice as high as mode 0. It can print 10 lines of 20 characters each (2 more lines are blocked by the text window) in 4 different colors. Its default colors are the same as mode 1. In fact, mode 2 is identical to mode 1 except that the characters are twice as high.

MODE 3

The first of the graphics modes, mode 3, like modes 1 and 2 has a text window; but instead of characters in the graphics window, it presents graphics pixels. These pixels are large: there are 20 rows of 40 pixels each (a 20 × 40 display area, or 800 pixels). It is a low-resolution mode. The pixels can be displayed in four different colors, including the background color. The default colors are black background, orange, green, and blue.

MODE 4

This graphics mode has a text window. The pixels are smaller than those in mode 3: there are 40 lines of 80 pixels each (3200 pixels altogether). In this mode there are only two colors for pixel display,

including the background color, making it a two-color mode, as opposed to mode 3, which is a four-color mode. Its default colors are black background and orange.

MODE 5

Mode 5 is the same as mode 4, except that it is a four-color mode. Its default colors are black background, orange, green, and blue.

MODE 6

A graphics mode with a text window, mode 6 has pixels that are even smaller than those in modes 4 and 5. There are 80 lines of 160 pixels each (12,800 pixels altogether). Like mode 4, this is a two-color mode. Its default colors are black background and orange.

MODE 7

Mode 7 is a four-color version of mode 6. Its default colors are black background, orange, green, and blue.

MODE 8

This extremely high-resolution graphics mode has the tiniest pixels available on the Atari. It has a text window. There are 160 rows of 320 pixels each. This mode uses only one color displayed in two different luminances. Its default colors are blue background and white (really a very light blue).

For easy reference, refer to Appendix A on p. 36 which contains a comparison chart of all Atari text and graphics modes.

Now that we know the modes, we use the GRAPHICS command to select the one we wish. GRAPHICS *m*, where *m* is our mode number, will put the computer into the mode desired. To get back to default mode 0, either hit SYSTEM RESET or enter GRAPHICS 0. This will clear the screen as well.

If you wish to eliminate the text window from any mode, simply add 16 to the mode number. For example, GRAPHICS 18 (2 + 16) gives us mode 2 without a text window. Eliminating the text window gives you room for more lines of modes 1 and 2 text and, in modes 3 through 8, more graphics pixels. One word of caution, though: as soon as the Atari finishes a program in these windowless modes, it reverts back to mode 0 to display the READY prompt. This can happen so quickly that you see only a small flash of the new mode before mode 0 takes over again. To circumvent this, you can use an endless loop at the program's end to keep it displaying the mode. For example:

PROGRAM 2

```
10 GRAPHICS 18
20 PRINT #6;"FLASH"
30 GOTO 30
```

First try program 2 without line 30; then add line 30. To escape the loop, hit BREAK or SYSTEM RESET. Line 30 can also use a FOR/NEXT loop,

which will keep the mode 18 display for a set amount of time, and then return to mode 0. Try substituting:

```
30 FOR I = 1 TO 500: NEXT I
```

Each time we use the GRAPHICS command, it clears the screen and restores the default colors. In fact, using GRAPHICS 0 in a program is an easy way to wipe the screen clean for later text display. If you wish to avoid clearing the screen when changing modes, add 32 to the mode number. Use this with care, though, because it tends to display garbage when used with modes 5 through 8, and may not keep the desired part of the screen when going from high-resolution to low-resolution modes.

SETCOLOR

Earlier we talked about color registers, where colors were loaded for later reference. Using our artist analogy, the color registers are equivalent to the palette used, the colors we have selected. Unlike the artist, though, we cannot mix our colors. They must remain separate within their color registers.

We have 5 color registers. They are numbered 0 through 4, and are loaded with default colors when we turn on the computer or hit SYSTEM RESET. If we wish to put in another color, we type SETCOLOR *r,h,l,* where *r* is the number (from 0 to 4) of the color register we want to load, *h* is the hue we want to put in (0 to 15), and *l* is the luminance (brightness) of the color we want (even numbers from 0 to 14). The hue chart, Plate 4 on the inside back cover, shows you the 16 hues by number. Luminance is determined by the size of the *even* number used. The number 0 gives you the darkest luminance, 14 the lightest.

When we load a new color into a color register, every pixel affected by that register will immediately change color. For example, in mode 0, let's change colors in register no. 2, which controls the background color. Enter:

```
SETCOLOR 2,4,0
```

The 2 selects the register, 4 selects pink, and 0 makes the pink as dark as possible. This should produce quite a change on your screen! To get back to blue, hit SYSTEM RESET or enter GRAPHICS 0.

Remember that the color registers are already loaded with default colors, so we only need to use the SETCOLOR command when we wish to change the colors.

COLOR

This command tells later input/output commands what number to load into individual screen pixels. It is usually used for the graphics modes, in which each number stored in a pixel calls up a separate color register. The number 1 in a pixel brings up register no. 0. Number 2 brings up register no. 1, and number 3 brings up register no. 2. To confuse the situation a

little more, the number 0 brings up register no. 4. For clarification, Figure 10 gives a small chart. Register no. 3 is only used in text modes 1 and 2.

In graphics modes, this number in a pixel:	brings up this color register:
1	#0
2	#1
3	#2
0	#4

FIGURE 10. Color register chart.

The form of the COLOR command is: COLOR *n*, where *n* is the number to be placed into the pixel. It is always used in conjunction with an input/output command.

INPUT/OUTPUT COMMANDS

PLOT

The PLOT command places the invisible graphics cursor at one pixel and loads that pixel with the number given in the last COLOR command. The way we use it is PLOT *x,y*, where *x* is the column number of the pixel and *y* is the row number. It is usually used in graphics modes but can also be used in text modes. The number loaded in the pixel brings up a color in graphics modes and a character in text modes. For example:

PROGRAM 3

```
10 GRAPHICS 3
20 COLOR 3
30 PLOT 10,5
```

This program should put a blue square (one pixel) on the screen, 11 columns over and 6 rows down. The command COLOR 3 brings up color register #2 (default color is blue), and PLOT 10,5 places this into the pixel. If you enter:

```
10 GRAPHICS 2
```

and run the program, you should see a green "#" on the screen 11 columns over and 6 rows down. By changing to mode 2, the 3 we loaded into the pixel is now read as the ATASCII character "#" in color register 1.

DRAWTO

This command draws a line from the last cursor position to the new pixel address, filling in pixels along the way with the number given in the last

COLOR command. Its format is: DRAWTO *x,y*, where *x,y* is the new pixel address. For example, add this to program 3 (in mode 3):

```
40 DRAWTO 0,0
```

and run the program. You should see a line of pixels, from address 10,5 to address 0,0, drawn as straight as possible.

The cursor has been left on the new pixel address given in the DRAWTO command.

POSITION

Technically, this is not an input/output command, because it does not change or read numbers from a pixel. Nonetheless, it is closely tied to the PRINT, PUT, and GET commands. POSITION *x,y* will move the invisible graphics cursor (or the mode 0 cursor) to the pixel address given by *x,y*. It will *not* change or read any numbers in pixels along the way or at the new address.

PRINT

This command is probably very familiar to you, but it has graphics uses that you may not yet know. The PRINT command is followed by a string (in quotation marks), a variable, or an arithmetic expression. This command evaluates the expressions, finds current values for variables, and then sends a sequence of ATASCII code numbers to the screen or to an outside device. These ATASCII code numbers can then be translated to their appropriate characters.

How do we control where the PRINT statement sends its ATASCII code? If we use an unmodified PRINT command, it will automatically send its code to the screen, where it will display it in mode-0 textual form. In modes 1 through 8, it will display in the text window. If we use modes 1 through 8 without a text window, the PRINT statement will break us out of that mode, return to mode 0, and clear the screen to display its message.

To send the ATASCII code elsewhere, we follow the PRINT command with a *device number*. This number, a number from 0 to 7, refers the Atari to an outside device (e.g., printer, disc drive). For more information on device numbers, read Chapter 5 in the *Atari BASIC Reference Manual* (published in 1980 by Atari, Inc). This manual should be included with your Atari 400 or 800 computer. In this Handy Guide, the only device number we will be concerned with is no. 6.

Device no. 6 is automatically set to be the graphics window of the television screen when we are in modes other than 0. By using the command PRINT #6; (the # and ; are essential), we cause the ATASCII code to be sent to the graphics window in modes 1 through 8. (Later we will see how to use this modified print statement.)

In mode 0, when we enter:

```
PRINT "I'M SENDING ATASCII CODE TO THE
SCREEN."
```

the Atari evaluates each character within the quotation marks and converts it to the appropriate ATASCII numeral. As it does so, it sends that numeral to the cursor, which inserts it into the pixel on which it is resting. The cursor then moves one address to the right, the next ATASCII numeral is sent, it is inserted into the pixel, and so on until the entire string is displayed.

One important aspect of the PRINT command is cursor location. As you see, once in operation it moves the cursor one pixel at a time, left to right. When it reaches the right edge of the screen, it drops one row down and returns to the far left of the screen (just as we do when reading or writing). How does it know where to start? It starts wherever the cursor is positioned. Where does it leave the cursor when it is done? If the last character following the PRINT command is a ,(comma) and is not enclosed in quotation marks, the cursor comes to rest at the next column-stop position. (Column-stop positions are preset in the Atari; they are not the tabs set by the keyboard.) If the last character is a ;(semicolon) and is not enclosed in quotation marks, then the cursor rests at the pixel to the right of the last pixel filled with an ATASCII numeral. If neither the comma nor the semicolon is used, the cursor is sent one row down, to the far left of the screen (the beginning of the next line). For example:

```
THIS IS WHERE THE CURSOR RESTS AFTER A
COMMA,                 □


THIS IS WHERE THE CURSOR RESTS AFTER A
SEMICOLON; □


THIS IS WHERE THE CURSOR RESTS WITH NO
SEMICOLON OR COMMA
□
```

Since the PRINT command starts wherever the cursor is positioned, we can use the POSITION command to start the PRINT wherever we wish. Entering:

```
GRAPHICS 0: POSITION 18,12: PRINT
"CENTER"
```

should clear the screen and then put CENTER in the middle of it. If we position the cursor to start the PRINT where something is already displayed, it will simply write over it. (Remember that when it loads each pixel with an ATASCII numeral, it erases what was previously there.)

USING PRINT IN TEXT MODES

Using the PRINT command in mode 0 is fairly simple, and you have probably had quite a bit of experience using it that way. However, when we use the

PRINT #6; command in modes 1 to 8, new graphics possibilities open up.

As we learned earlier, in modes 1 and 2 we are limited to 64 characters with 4 colors. These 64 characters are the ones we use most often: capital letters, numerals, punctuation marks, and other symbols such as dollar signs and asterisks. They correspond to ATASCII numbers 32 through 95 (see Appendix B on p. 41). The characters left out are small letters, graphics symbols, and all inverse characters.

When using the PRINT #6; command in text modes, uppercase letters in the quotes following will be reproduced on the screen as uppercase letters in the color controlled by color register 0. Lowercase letters will appear as uppercase letters controlled by color register 1. Inverse uppercase letters are displayed as color-register 2 uppercase letters, and inverse lowercase letters as color-register 3 uppercase letters. More succinctly: when you wish to print uppercase letters in modes 1 and 2, enter them after PRINT #6; as:

- Normal for the color of register 0 yellow orange
- Lowercase for the color of register 1 green
- Inverse for the color of register 2 blue
- Inverse lowercase for the color of register 3 Pinke

The same holds true for numerals and punctuation, but since there are no lowercase numerals and punctuation, only color registers 0 and 2 are available using this method. The graphic symbols and their inverses are used for registers 1 and 3. Appendix B correlates all the ATASCII numbers with their mode 0 and mode 1 or 2 characters.

It is possible to change our 64-character set displayed in modes 1 and 2 from numerals, punctuation, and uppercase letters to lowercase letters and graphics characters, by entering:

```
POKE 756,226
```

One problem with this is that the "space" character is not included in this character set, so a heart-shaped graphics symbol will fill the screen where spaces usually are. To return to the default character set, hit SYSTEM RESET or use the command:

```
POKE 756,224
```

Cursor positioning in these modes using the PRINT #6; command is controlled in the same way as is the PRINT command in mode 0—with commas, semicolons, or absence of punctuation. The main difference is that modes 1 and 2 will not *scroll* the screen for you. Scrolling takes place in mode 0 and in the text windows of other modes when you run out of room at the bottom of the screen to print. The Atari then shifts the entire contents of the screen up by one logical line, leaving room at the bottom for more text. In modes 1 and 2, the cursor will run off the bottom of the screen, giving you an error no. 141: cursor out of range. In these modes, then, you must plan carefully how many lines you will print.

USING PRINT IN GRAPHICS MODES

Up to now, we have used the PRINT #6; command as a way to display text. In modes 3 through 8, it can be used to color in graphics pixels. Recall that text modes use numbers from 0 to 255 in each pixel to call up different characters from the character set. Four-color graphics modes use a number from 0 to 3 to refer to color registers, and two-color modes use 0 or 1. What happens in four-color modes when the PRINT #6; command starts loading numbers larger than 3 into graphics pixels? The Atari simply chops it down to size. It divides the number by 4 (using integer division) and puts the remainder into the pixel. For example, if the cursor tries to load 65 (ATASCII for A) into a pixel, it is divided by 4, which gives 16 with a remainder of 1. The 16 is discarded, and the pixel is loaded with 1. This corresponds to COLOR 1, and will call up color register no. 0. In two-color modes, the Atari will divide by 2 and load the remainder, a 0 or a 1.

How do we use this tool? By printing strings of characters in graphics modes, we can fill in rows of pixels in many different colors without using tedious COLOR, PLOT, and DRAWTO commands. The character 0 will yield background color, the characters 1, 2, and 3 will yield colors 1, 2, and 3 (calling up registers nos. 0, 1, and 2). By combining POSITION and PRINT #6; we can print colored rows wherever we like. Try this:

```
GRAPHICS 3: POSITION 17,10: PRINT #6;
"123123"
```

A three-color bar should appear in the middle of the screen.

We can print several strings in a column that will combine to create complex patterns and pictures. Experimentation and imagination will guide you from here.

LOCATE

The simple command LOCATE is used to position the cursor at a pixel address and read the contents of the pixel. It works in all modes. To use it, we enter LOCATE *x,y,*variable, where *x* and *y* give the pixel address and the variable sets up a place to store the contents of the pixel. For example, if we want to look at the contents of pixel 14,18 we enter:

```
LOCATE 14,18,P: PRINT P
```

The Atari then prints out the contents of pixel 14,18—a number from 0 to 255 in the text modes, 0 to 3 in the four-color graphics modes, and 0 to 1 in the two-color modes. The cursor remains on address 14,18.

Although this command is used only to read the contents of a pixel without changing it, following the LOCATE command with a PRINT may alter the contents of the pixel. Be prepared to reload the pixel.

PUT and GET are two commands that access individual pixels. The PUT command reads the number that a particular pixel holds. When we use these two commands we have to specify a device number, as we did in the modified PRINT statement. For graphics usage, we again use #6 to specify the graphics window of the TV screen. Here are the formats to use: PUT #6,*n*, where #6 is the device number and *n* is the number to load into the pixel; GET #6, variable where #6 is the device number and the variable (X, A, J, PIX, whatever you wish to use) receives the number held in the pixel.

Isolated PUT and GET statements work much the same as PLOT and LOCATE commands. Entering:

```
GRAPHICS 3: POSITION 5,10: PUT #6,3
```

will plot a blue pixel at address 5,10. Entering:

```
GRAPHICS 3: POSITION 5,10: GET #6,X:
PRINT X
```

will print a 3, the contents of pixel 5, 10, in the graphics window.

The important difference between PUT/GET and PLOT/LOCATE lies in cursor positioning. PLOT and LOCATE commands need an address for the cursor each time they are used. The PUT and GET commands do not use an address but instead work on whatever pixel the cursor is positioned on. This is why we used a POSITION command in the examples above. Even more important, after a PLOT or LOCATE command, the cursor remains on the same pixel. Following a PUT or a GET, the cursor automatically moves one pixel to the right. If it is already at the far righthand edge of the screen, the cursor drops one row and jumps to the far left, as the PRINT command does. This is important, because consecutive PUT or GET commands can be used to read or input long rows and large blocks of pixels without tediously reentering new addresses for each command.

To see how this works, as well as to demonstrate other familiar graphics commands and concepts, enter and run Program 4.

This program will draw lines in mode 3 (Figure 11A), will then use a loop of GET statements to gather the display numerals from each pixel in the graphics window, will print out the results in mode 0 (Figure 11B), and will conclude by returning to mode 3 and reloading the lines using a loop of PUT statements. We will use array A with 801 (0 to 800) elements to store the contents of the 800 pixels in mode 3.

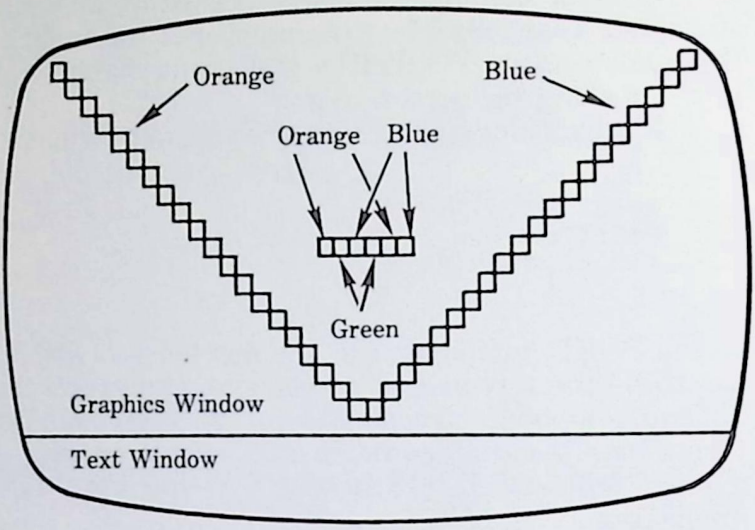
PROGRAM 4

```
10 DIM A(800)
20 POKE 82,0: POKE 83,39
30 GRAPHICS 3
40 COLOR 1: PLOT 0,0: DRAWTO 19,19
50 COLOR 3: PLOT 39,0: DRAWTO 20,19
60 POSITION 17,10: PRINT #6; "123123"
100 POSITION 0,0
110 FOR I=1 TO 800
120 GET #6,X
130 A(I)=X
140 NEXT I
200 GRAPHICS 0
210 FOR I=1 TO 800
220 PRINT A(I);
230 NEXT I
240 FOR J=1 TO 2000: NEXT J
300 GRAPHICS 3
310 POSITION 0,0
320 FOR I=1 TO 800
330 PUT #6, A(I)
340 NEXT I
```

Let us analyze this program line by line.

- 10—Dimension array A.
- 20—These two POKEs set the margins all the way to the edge in mode 0. (We will discuss this in Chapter 5.)
- 30—Go to mode 3.
- 40—Draw an orange line.
- 50—Draw a blue line.
- 60—Draw a multicolored line.
- 100—Put the cursor in the upper lefthand corner to begin reading the pixels.
- 110—Set up an 800-step loop.
- 120—Read the current pixel, pop the value into X, move the cursor to the next pixel.
- 130—Load the value of X into an ordered element of array A.
- 140—Complete the loop.
- 200—Go back to text mode 0.
- 210—Set up an 800-step loop.
- 220—Print out each element of the array A in consecutive order.
- 230—Complete the loop.
- 240—Kill some time so the viewer can look at the display.
- 300—Return to mode 3.
- 310—Put the cursor in the upper lefthand corner to begin loading the pixels.
- 320—Set up an 800-step loop.
- 330—Fill each pixel.
- 340—Complete the loop.

Mode 3 (Graphics mode)



Mode 0 (Text mode)

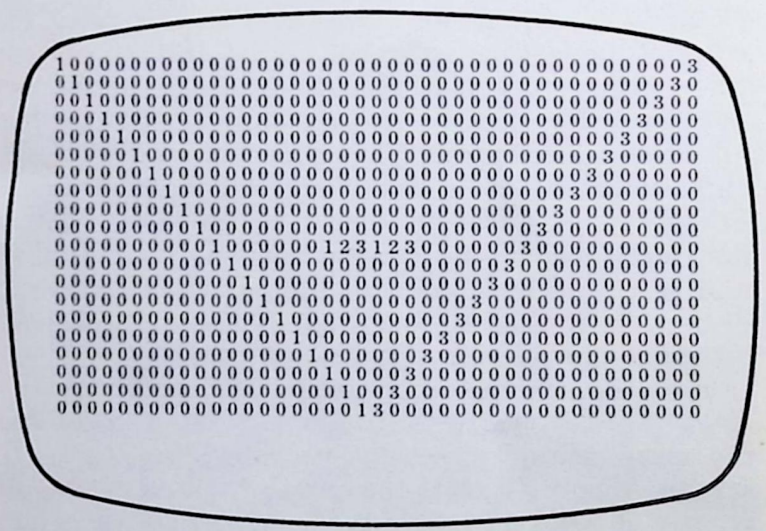


FIGURE 11 A & B. Program 4 in mode 3 and mode 0.

Some things to notice: in lines 40 and 50, we used the PLOT and DRAWTO method to draw our lines, while in line 60 we used the PRINT method. Whenever we read or load individual pixels, especially when there are 800 of them, it takes time. This is why it takes more time to display in mode 3 using the loop in lines 300 to 340 than it does using lines 40 through 60, which can ignore blank pixels. The display in mode 0 (lines 200 to 240) should give a very clear idea of how the pixels are loaded in mode 3. The pixels occupy the same locations in these two modes, so the patterns show up in both.

Obviously, saving and loading graphics windows is only one use for PUT and GET commands. You will discover many more.

XIO

The XIO command is a tricky and potentially confusing command, but it can save us a lot of programming trouble if we learn to use it. Its purpose in graphics is to fill in areas of the screen with one color or character quickly and easily.

To work effectively, XIO should be used with several other commands. It is preceded by a POKE statement that tells XIO what number to use to fill in the screen and a POSITION statement that tells it how far along the screen to go.

For graphics applications, the program format is:

```
POKE 765,n
POSITION x,y
XIO 18,#6,0,0,"S:"
```

In the POKE command, *n* is the number you wish to use to fill the screen-area pixels, and *x* and *y* are the XIO ending boundary address. The XIO command format has no variables and is used just as you see it—always followed by 18,#6,0,0,"S:"—for the "fill" application.

Now for the specifics of XIO behavior. XIO does several things at once. It draws a line from the last pixel plotted to the new cursor position given with the POSITION command. It draws this line using the color of the last plotted pixel; that is, if that pixel was filled with color 1, it will plot all points of the new line in color 1. For each pixel it plots (up or down), it fills all background pixels (those loaded with 0) to the right with the color number given in the previous POKE 765,*x* statement. It does this until it hits a non-zero pixel. As soon as this happens, it stops filling and plots the next point of the boundary line being drawn. If it fills all the way to the right without hitting a non-zero pixel, it wraps around to the far left side of the screen (on the same line) and continues filling until it reaches a non-zero pixel. You can see the wrap-around effect in program 5, on the lower screen. When XIO hits the cursor position given previously, it plots the end of the boundary line there and fills to the right. It leaves the cursor at the last pixel of the fill line.

To see just how the XIO command works, let's enter and analyze this program:

PROGRAM 5

```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 110, 60
40 DRAWTO 130, 40
50 DRAWTO 110, 20
60 DRAWTO 50,20
70 POKE 765,3
80 POSITION 40,70
90 XIO 18,#6,0,0,"S:"
```

dead

get yel
Horiz line
don't know
why yellow

55 STOP

change values stop where
to fill in

try with #1 - works

If you run this program, you will see three sides of a figure drawn in orange (the two right sides, and the top side). As the left side is drawn, also in orange, the figure is filled from the left to the right side with blue. As soon as the right hand boundary of the figure is passed going down, the blue filler covers the entire width of the screen, wrapping around to fill in even to the far left portion of the left side. All this was achieved as follows:

- 10—Enter mode 7.

- 20—Select color 1 (color register 0) for use (orange).
- 30—Plot a point at the lower right side of the graphics window.
- 40—Draw a line from the previous point up and to the right.
- 50—Draw a line from the end of the last one up and to the left.
- 60—Draw a line from the last one straight to the left.
- 70—Tell the XIO command to use color 3 (~~color~~ register 2) to fill in the figure (blue).
- 80—Move the cursor (without drawing) to the lower left side of the screen.
- 90—Draw a line in orange (color 1) from the last plotted point to the new cursor position. As each point is plotted, fill in every background pixel to the right with blue (color 3) until a non-background pixel is reached.

A couple of points to consider: XIO will only fill in areas of background color (zero pixels). It will not cover over non-background colors. Beware: if for some strange reason you are drawing *and* filling with color 0, when XIO fills it may not encounter a non-zero pixel and will enter an endless loop!

COMMAND SUMMARY

We have now examined the Atari BASIC graphics commands and their functions. The formatting commands allow us to prepare for later pixel changes on the screen:

- GRAPHICS determines the mode we use
- SETCOLOR loads a color register with the color we want
- COLOR determines the color to be used for the drawing commands that follow it.

The input/output commands let us display colors and characters on the screen:

- PLOT and DRAWTO fill specified pixels with a previously selected color or character
- PRINT fills successive pixels with a string of characters or colors
- POSITION moves the cursor around the screen without changing anything
- LOCATE examines the contents of any individually addressed pixel
- PUT successively loads pixels with colors or characters
- GET successively examines the contents of pixels
- XIO fills in enclosed figures with a color or characters.

Using these commands in a BASIC program can provide us with effective graphics displays.

4. THE GTIA CHIP AND GRAPHICS MODES 9, 10, AND 11

Three new graphics modes are accessible through Atari BASIC on most Atari computers sold after January of 1982. These computers contain a GTIA video display chip that replaces the less versatile CTIA chip used in earlier Ataris. The video-display chip is the part of the Atari computer system that displays data on the screen. The way it displays depends on which graphics mode has been selected.

Since the Atari computer system was designed to use the GTIA chip, it is very easy to update any computer with an older CTIA chip. The GTIA is available at most Atari service centers, where it can be installed. If you are familiar with the innards of the Atari, you can install it yourself in about half an hour. If you decide to do it yourself, make sure you replace the proper chip and position the GTIA chip in the right way: otherwise you will damage the GTIA and possibly your computer.

One immediate benefit of the GTIA chip is enhanced color on the TV screen. The most important advantage of the GTIA over the CTIA is that three new graphics modes are added to Atari BASIC: mode 9, which has one hue and sixteen luminances; mode 10, which has nine separate colors; and mode 11, which has one luminance and sixteen hues.

These new modes (called *GTIA modes*) have several features in common. One is the pixel size. It is very small and allows very high-resolution graphics. Each pixel is the height of a mode 8 pixel but is four times as wide (Figure 12). It looks like a tiny rectangle rather than the square pixels in modes 3 through 8. There are 80 columns and 192 rows in modes 9–11.

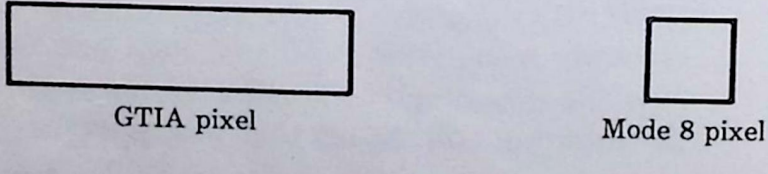


FIGURE 12. GTIA and mode 8 pixels.

Another feature is that no text window exists in the GTIA modes. Adding 16 to the mode number after a GRAPHICS command does not change this. Adding 32 does work, by not clearing the screen after a mode change.

To see the new modes, enter and run the following program. (If your computer has only a CTIA chip, this program will be very disappointing.)

PROGRAM 6

```
10 REM gtia mode demo
30 GRAPHICS 0: POKE 752,1: POSITION 2,7
40 ? "THIS IS A DEMONSTRATION OF MODES
9-11."
50 ? "TO GO FROM ONE DISPLAY TO THE
NEXT,"
```

D: GRAPHICS.2

```
60 ? "PRESS THE start BUTTON AFTER EACH"
70 ? "DISPLAY IS DONE."
80 ? :? "TO SEE THE LUMINANCES AVAILABLE
IN"
90 ? "MODE 9, PRESS start.  TO CHANGE"
100 ? "THE HUE BASE, PRESS start
AGAIN....."
110 DEMO=0
120 IF PEEK(53279)=6 THEN 140
130 GOTO 120
140 DEMO=DEMO+1
150 ON DEMO GOSUB 1000, 2000, 3000,
4000, 5000, 6000, 7000, 8000, 9000, 170
160 GOTO 120
170 END
1000 REM mode 9 luminances
1010 GRAPHICS 9
1020 SETCOLOR 4,0,0
1030 FOR H=0 TO 15
1040 COLOR H
1050 FOR P=0 TO 4
1060 PLOT 5*H+P,0: DRAWTO 5*H+P,191
1070 NEXT P: NEXT H
1080 RETURN
2000 REM change mode 9 color base
2010 FOR C=1 TO 15
2020 SETCOLOR 4,C,0
2030 FOR T=1 TO 300: NEXT T
2040 NEXT C
2050 RETURN
3000 GRAPHICS 0: POKE 752,1: POSITION
1,12
3010 ? "THIS IS THE PIXEL SIZE FOR MODES
9-11:": FOR T=1 TO 100: NEXT T
3020 RETURN
4000 REM gtia pixel size demo
4010 GRAPHICS 11: SETCOLOR 4,0,10
4020 FOR J=1 TO 100
4030 X=INT(80*(RND(0))):
Y=INT(192*(RND(0))): C=INT(15*(RND(0)))+1
4040 COLOR C: PLOT X,Y
4050 NEXT J
4060 RETURN
4070 NEXT X
4080 NEXT Y
4090 RETURN
5000 GRAPHICS 0: POKE 752,1: POSITION
2,11
5010 ? "MODE 10 HAS THE SAME PIXEL SIZE,
BUT"
5020 ? "CAN SUPPORT UP TO 9 DIFFERENT
COLORS"
5030 ? "WITH INDEPENDENT HUES AND
LUMINANCES:": FOR T=1 TO 100: NEXT T
5040 RETURN
6000 REM mode 10 color demo
6010 GRAPHICS 10
6020 POKE 704,0: POKE 705,30: POKE
706,120: POKE 707,160
6030 SETCOLOR 0,4,4: SETCOLOR 1,3,12:
SETCOLOR 2,14,2: SETCOLOR 3,5,12:
SETCOLOR 4,1,2
6040 FOR C=1 TO 8
6050 COLOR C
6060 FOR X=0 TO 4
6070 PLOT 10*(C-1)+X,0: DRAWTO
10*(C-1)+X,191
6080 NEXT X
6090 NEXT C
6100 RETURN
7000 GRAPHICS 0: POKE 752,1: POSITION 2,9
7010 ? "MODE 11 HAS ONLY ONE LUMINANCE,
BUT"
7020 ? "CAN SUPPORT 16 DIFFERENT HUES."
7030 ? :? "PRESS start TO SEE THE HUES"
7040 ? "AVAILABLE, PRESS start AGAIN TO
SEE"
```

```
7050 ? "THE BASE LUMINANCE
CHANGE..........": FOR T=1 TO 100: NEXT T
7060 RETURN
8000 REM mode 11 hues
8010 GRAPHICS 11
8020 SETCOLOR 4,0,0
8030 FOR H=0 TO 15
8040 COLOR H
8050 FOR P=0 TO 4
8060 PLOT 5*H+P,0: DRAWTO 5*H+P,191
8070 NEXT P: NEXT H
8080 RETURN
9000 REM change mode 11 luminance base
9010 FOR L=2 TO 14 STEP 2
9020 SETCOLOR 4,0,L
9030 FOR T=1 TO 200: NEXT T
9040 NEXT L
9050 RETURN
```

Now that we have seen the new modes, let's examine each one individually. We will start with mode 10, since it most closely resembles the CTIA graphics modes we worked with earlier.

MODE 10

The main difference between mode 10 and standard graphics modes (aside from pixel size) is that there are 9 color registers to work with instead of 4 or 2. There is one problem with this: we can only access 5 color registers using the SETCOLOR command. Fortunately, there is an easy solution.

When we command the computer to SETCOLOR *r,h,l,* it finds the color register location in its memory. Then it multiplies the hue value by 16, adds the luminance value, and stores this value in the appropriate memory address. It has now changed the color register to the color for which we asked.

We can do this same operation ourselves. Instead of commanding SETCOLOR 0,4,6, for example (which puts red into register no. 0), we can POKE the number directly into the color register. To do this we must know the memory location of color register no. 0; it is memory address 708. Then we convert the hue and luminance values into one number as the computer did above: hue times 16 plus luminance—in this case, $(4 \times 16) + 6$, which equals 70. Now we poke this value into 708: POKE 708,70. We have just changed color register no. 0 without using the SETCOLOR command.

To summarize the POKE method of changing color registers:

1. Convert hue and luminance to an even number from 0 to 254 using the formula $(h \times 16) + l$ where h = hue and l = luminance.
2. POKE the resultant value into the memory address of the color register you wish to change. In mode 10 there are 9 registers, in memory locations 704 through 712.

Now we can load the color registers that are not accessible through the SETCOLOR command. These are in memory locations 704 through 707. Memory locations 708 through 712 correspond to SETCOLOR numbers 0 through 4. You can set the colors here by using the SETCOLOR command or by the POKE method, whichever you find more convenient. (For an example of register setting, see lines 6020 and 6030 in Program 6.)

Plotting pixels on the screen works the same way in mode 10 as in previous graphics modes, except that the range of values we insert into the pixels is greater. We insert values from 0 through 8, which correspond to the color registers in memory addresses 704 through 712. To do this, we can use the COLOR, PLOT, and DRAWTO commands or we can use the PRINT #6; method. The value 0 in a pixel calls the background-color register, which is in memory address 704. Values 1 through 8 call up the colors in registers located in addresses 704 through 712. Default colors are black in memory addresses 704–707 and 712, orange in 708, light green in 709, dark blue in 710, and red in 711. The chart in Figure 13 should make this clearer.

Color #:	Calls up register in memory location #:	Which is SETCOLOR register #:	Loaded with default color:
0	704	—	black
1	705	—	black
2	706	—	black
3	707	—	black
4	708	0	orange
5	709	1	light green
6	710	2	dark blue
7	711	3	red
8	712	4	black

FIGURE 13. Chart of mode 10 pixel value-color register equivalencies.

The advantages of this mode are clear. It is very easy to create high-resolution graphics displays with many colors. By rotating colors from register to register it is possible to achieve the illusion of motion on the screen. You can see this by entering and running Program 7.

PROGRAM 7

```
10 REM Zigzag
20 REM a mode 10 demo
50 GRAPHICS 10
60 FOR R=704 TO 712: READ C: POKE R,C:
NEXT R
70 DATA 0,148,182,216,30,58,74,68,100
80 DIM S$(16): I=-33
90 FOR B=1 TO 3
100 S$="ABCDEFGHABCDEFGH": T=1:
I=I+32: GOSUB 160
110 S$="HGFEDCBAHGFEDCBA": T=2:
I=I+32: GOSUB 160
120 NEXT B
130 X=PEEK(712)
140 FOR R=1 TO 8: Y=PEEK(704+R): POKE
704+R,X: X=Y: NEXT R: IF PEEK(53279)=6
THEN END
150 GOTO 140
160 FOR Y=1 TO 16
170 POSITION 0,I+Y
180 FOR X=1 TO 10
190 ON T GOSUB 280,290
200 NEXT X: NEXT Y
```

D:Zigzag

```
210 FOR Y2=16 TO 1 STEP -1
220 POSITION 0,I+16+Y2
230 FOR X=1 TO 10
240 Y=17-Y2
250 ON T GOSUB 280,290
260 NEXT X: NEXT Y2
270 RETURN
280 PRINT#6;
S$(9-(Y-(INT((Y-1)/8)*8)),
16-(Y-(INT((Y-1)/8)*8)));: RETURN
290 PRINT#6;
S$(1+(Y-(INT((Y-1)/8)*8)),
8+(Y-(INT((Y-1)/8)*8)));: RETURN
```

MODE 9

This mode has only one hue, but it is available in 16 different luminances. This gives us a capability of delicate shading that is not possible in other modes.

Since there is only one hue used (called the *base hue*), we need only use one color register, register no. 4. We choose the base hue we wish to use, a value from 0 through 15, and load it in register no. 4 using a SETCOLOR command. We set the luminance to 0. If we use a higher luminance value, we will be restricted to fewer luminances when we plot pixels later on. For example, to set our base hue to orange, we use the command SETCOLOR 4,2,0.

Plotting the pixels is easy. We load them with values from 0 through 15: the higher the value, the higher the luminance of the displayed pixel. As before, we can use the COLOR, PLOT, and DRAWTO commands to do this or we can use the PRINT #6; method. Since there are 16 different values to present, useful characters in using the PRINT #6; method are 0 for luminance 0, and A through O (letter O, not zero) for luminances 1 through 15.

The default color in mode 9 is black. This means register no. 4 is set to 0,0.

The delicate shading available in mode 9 is particularly useful for presenting three-dimensional images. By entering and running Program 8, you can see an example of this shading. (See Plate 2 on inside front cover.)

PROGRAM 8

```
10 REM Sinewave
20 REM mode 9 picture demo
40 GRAPHICS 9
50 SETCOLOR 4,3,0
60 DEG
70 FOR X=0 TO 79
80 COLOR ABS(INT(15*SIN(9*X)))
90 PLOT X,20-(INT(19*SIN(9*X)))
100 DRAWTO X,160-(INT(19*SIN(9*X)))
110 COLOR 5
120 PLOT X,161-(INT(19*SIN(9*X)));
DRAWTO X,191
130 NEXT X
140 GOTO 140
```

D:Scroll

also makes in GR10 full color

MODE 11

Mode 11 is the reverse of mode 9: there is only one luminance, with 16 hues. This allows us vivid multicolored displays. As in mode 9, we use only one color register, this time to set the base luminance. As before, this is register no. 4. We choose the luminance

we wish to use, and insert the value (an even number from 0 to 14) into register no. 4. We set the hue value to 0, for otherwise the number of colors available to us while plotting pixels will be limited.

Plotting the pixels also works very much as it does in mode 9. We load the pixels with values of 0 through 15. The values in mode 11 correspond to the 16 hues we have been using (see Plate 4 on inside back cover). Now we can see them all at once if we wish. We can use the COLOR, PLOT, DRAWTO, and PRINT #6; methods to load the pixels. One item to note: a pixel value of 0, which is the background color of black, is not affected by the base luminance. It is always at luminance 0. This gives a good black border and background no matter what base luminance we are using. The default setting in mode 11 is luminance 6: this means that register no. 4 is set to 0,6.

Mode 11 is best used to present a rainbow spectrum of colors on the screen at one time. Program 9, the mode 11 demonstration program shown below, demonstrates this capability. (See Plate 3 on inside front cover.)

PROGRAM 9

```
10 REM Popsicle
20 REM a mode 11 demo
40 GRAPHICS 11
50 SETCOLOR 4,0,10
60 FOR Y=0 TO 5
70 COLOR 3: READ X,W: PLOT X,Y: DRAWTO
W,Y
80 NEXT Y
90 DATA
36,43,33,46,30,49,28,51,26,53,25,54
100 X=24: W=55
110 FOR Y=6 TO 11
120 PLOT X,Y: DRAWTO W,Y
130 NEXT Y
140 FOR A=1 TO 9:READ C
150 FOR Y=A*12-1 TO A*12+11
160 COLOR C: PLOT X,Y: DRAWTO W,Y
170 NEXT Y: NEXT A
180 DATA 4,5,6,7,8,9,10,12,13
190 FOR Y=120 TO 133
200 PLOT X,Y: DRAWTO W,Y
210 NEXT Y
220 FOR Y=134 TO 135
230 READ X,W: PLOT X,Y: DRAWTO W,Y
240 NEXT Y
250 DATA 25,54,26,53
260 FOR Y=136 TO 189
270 COLOR 2: PLOT 37,Y: DRAWTO 42,Y
280 NEXT Y
290 PLOT 38,190: DRAWTO 41,190
300 PLOT 39,191: PLOT 40,191
310 GOTO 310
```

D'ICECREAM

The GTIA chip offers us a considerable advantage in creating high-resolution displays in many different colors and shades, and its features are easily used with BASIC commands. It is definitely worthwhile to learn and use the new GTIA modes whenever possible.

5. TIPS AND TRICKS

This last section is a small collection of odds and ends to make graphics programming a little neater and easier. This is by no means a comprehensive collection—it is just the tip of the iceberg. You will no doubt collect many more as you program. It would be a good idea to keep a record of them for easy reference.

1. Making the graphics cursor invisible in mode 0: POKE 752,1 will turn the cursor off; POKE 752,0 will turn it back on.

2. Clearing the screen while running a program: a PRINT"◥"command will clear the screen. (To get this character, hit the ESC key, then hit SHIFT and CLEAR simultaneously.) To clear the graphics window in modes 1 through 8, use PRINT #6;"◥".

3. Keeping the screen out of "attract" mode: use POKE 77,0 at least once every 7 minutes in your program. Your Atari will begin to shift colors on the screen to protect it from burnout if it detects no keyboard activity for longer than approximately 9 minutes. This can be annoying if you are using joysticks or if the computer is drawing designs slowly without user input. POKE 77,0 resets the attract mode timer to 0.

4. Freezing a graphics display: hit the 1 key while holding down the CONTROL key. Doing this again will unfreeze the display and allow it to continue. This is especially handy when a long program is scrolling past you in mode 0. You can stop and start it at any time without hitting BREAK and starting over again. CONTROL 1 only interrupts the graphics display and will not stop non-display commands in process.

5. Setting the margins in mode 0: POKEing 82 will set the left margin; POKEing 83 will set the right margin. The number you POKE into locations 82 and 83 should be the number of the column at which you want the margin. POKE 82,0 sets the left margin at the left edge of the screen. POKE 83,39 sets the right margin at the right edge of the screen.

6. Using graphics characters: If you press down the CONTROL key while typing letters of the alphabet, you will notice strange characters appearing on the screen. These are graphics characters and can be used to create pictures and charts in mode 0. There are blocks, half-blocks, diagonal lines, horizontal lines, and many other shapes which combine easily into recognizable images. There are also card suite symbols (clubs, hearts, spades, and diamonds), which make it easy to display card games. To type in graphics characters with ease, press the caps/lower key while holding down the CONTROL key.

This will lock the keyboard into the CONTROL mode. Pressing the caps/lower key while holding the SHIFT key will return your keyboard to the capital letter mode.

Just for fun: Enter the command POKE 755,4. Don't ask why, just try it! To get back to normal, enter the command POKE 755,2.

We have now finished our tour of video graphics using Atari BASIC. Although we have a wide range of video display effects available with what we have learned, we have just scratched the surface of the Atari graphics capabilities. Many advanced graphics techniques are available to a programmer with a good knowledge of the internal memory structure of the Atari computer and some assembly language. You must have seen many of them in the commercial programs available for the Atari. *Player/missile graphics* allow you to place an object on the screen that will move in response to joystick controls. *Display list modification* allows you to mix graphics modes on one screen. (This lets you use mode 1 and 2 characters as titles for graphics displays.) Designing your own *character sets* allows you to create your own graphics symbols. With *artifact coloring* you can get shades of colors not usually available on the Atari and put many more colors on the screen than your particular graphics mode usually allows. *Fine scrolling* allows you to scroll characters on the screen up, down, or sideways in smooth motion. Using *display list interrupts,* you can expand the power of all of the previous graphics techniques.

Although these graphics techniques require a more sophisticated knowledge of the Atari than the techniques we have learned in BASIC, they give you something to look forward to. By learning more and more about your computer, you gain increased facility and finesse for creating excellent video graphics.

We hope that this Handy Guide has provided you with the fundamental tools of graphics programming and a comprehensive concept of the Atari BASIC graphics system, so that you can now proceed with your own programming. It is up to you to apply imagination and structure to these tools in order to communicate effectively through video graphics. Good luck!

APPENDIX A: DISPLAY MODE INFORMATION

The following two charts are presented to help you keep track of color registers, pixel size, pixel load numbers, and other miscellaneous information that changes from mode to mode. The first chart deals with the text modes 0 through 2; the second chart presents the graphics modes 3 through 11.

TEXT MODES

1. "Number of columns" tells how many pixels there are horizontally across the graphics window.
2. "Number of rows" tells how many pixels there are vertically down the graphics window. The upper number is valid when a text window is present; the lower number is valid with no text window. (The text window is always in mode 0, with 40 columns and 4 rows.)
3. "Number of colors" tells us the maximum number of colors available in a particular mode.
4. "Default colors" shows us the colors automatically loaded into the color registers when the computer is turned on or SYSTEM RESET is hit. Each one is followed by its hue and luminance number as it would be entered after a SETCOLOR command.
5. "In color register #" shows where each of the default colors is loaded.
6. "Accessed in 'PRINT' by ATASCII characters" tells us what characters will access each color register. The numbers refer to the ATASCII code (in decimal) for each character. This is basically for modes 1 and 2. Color selection was discussed in the PRINT command section. Note that the background and border colors are not accessed by characters.
7. "RAM required (in bytes)" tells us how much memory will be needed to support a particular display mode. When the Atari enters the mode, it must take up this much memory to keep track of individual pixels. This information can be handy when writing a long program so that you can plan on less available memory when your program calls up a mode of higher resolution or more colors. Notice that two-color modes use much less memory than their four-color equivalents.

CHART 1. Text Modes

Mode	Number of columns (X axis)	Number of rows, w text window/ w/o text window (Y axis)	Number of colors	Default colors (hue and luminance)	In color register # (SetColorReg.)	Accessed in PRINT by ATASCII characters	RAM required (in bytes)
0	40	24	2, both of the same hue	light blue (9,10) dark blue (9,4) black (0,0)	1 (lum. only) 2 4	0 through 255 (all characters) background (no characters) border (no characters)	993
1	20	20 / 24	5	orange (2,8) light green (12,10) dark blue (9,4) red (4,6) black (0,0)	0 orange 1 green 2 blue 3 red 4 black	32–95 (uppercase letters and numerals) 0–31 and 96–127 (lowercase letters and graphics characters) 160–233 (inverse uppercase letters and numerals) 128–159 and 224–255 (inverse lowercase letters and graphics characters) background and border (no characters)	513
2	20	10 / 12	5	orange (2,8) light green (12,10) dark blue (9,4) red (4,6) black (0,0)	0 1 2 3 4	32–95 (uppercase letters and numerals) 0–31 and 96–127 (lowercase letters and graphics characters) 160–233 (inverse uppercase letters and numerals) 128–159 and 224–255 (inverse lowercase letters and graphics characters) background and border (no characters)	261

1. The number of columns, rows, and colors and the default colors mean the same in this chart as they do in the text-mode chart. However, for mode 11 there is no default color, just a default luminance.

2. "In color register #" is the same as in the text-mode chart, with the exception of mode 10. Here the registers are represented by their memory addresses (in parentheses), which follow their register number (if one exists).

3. "Accessed by pixel load (COLOR)#" shows what number in a pixel will access the color register of the previous column. This is the number we use in a "COLOR" command.

4. "Background color register #" and "border color register #" show which registers control the background and border.

5. "RAM required" is the same as in the text mode chart.

A word of explanation is needed for modes 0 and 8. Only the luminance differs: the hue for the background and the characters or graphics points is always the same; the background is controlled by register no. 2, and the characters or graphics points are controlled by register no. 1. This is true for the text window, too, so care must be taken to have a wide luminance difference between registers 1 and 2 when you reload registers for use in the graphics window.

Color Reg Set Color Reg
1 orange → 0
2 green → 1
3 blue → 2
0 black background 4

CHART 2. Graphics Modes

Mode	X Number of columns	Y Number of rows, w text window/ w/o text window	Number of colors	Default colors (hue and luminance)	Set Color Reg In color register #	Accessed by pixel load (COLOR #)	Background color register #	Border color register #	RAM required (in bytes)
3	40	20 / 24	4	orange (2,8) light green (12,10) dark blue (9,4) black (0,0)	0 1 2 4	1 2 3 0 (background and border)	4	4	273
4	80	40 / 48	2	orange (2,8) black (0,0)	0 4	1 0 (background and border)	4	4	537
5	80	40 / 48	4	orange (2,8) light green (12,10) dark blue (9,4) black (0,0)	0 1 2 4	1 2 3 0 (background and border)	4	4	1017
6	160	80 / 96	2	orange (2,8) black (0,0)	0 4	1 0 (background and border)	4	4	2025
7	160	80 / 96	4	orange (2,8) light green (12,10) dark blue (9,4) black (0,0)	0 1 2 4	1 2 3 0 (background and border)	4	4	3945

(Continued on next page)

CHART 2.: Graphics Modes (continued)

Mode	Number of columns	Number of rows, w text window/ w/o text window	Number of colors	Default colors (hue and luminance)	In color register #:	Accessed by pixel load (COLOR #):	Background color register #:	Border color register #:	RAM required (in bytes)
8	320	160 / 192	2, both of the same hue	light blue (9,10) dark blue (9,4) black (0,0)	1 (lum. only) 2 4	1 2 border (no pixels)	2	4	7900
9	80	192	1 hue, 16 luminances	black (0,0)	4	numbers 0–15 give 16 shades of luminance	4	4	7900
10	80	192	9	black (0,0) black (0,0); black (0,0) black (0,0); orange (2,8) light green (12,10); dark blue (9,4) red (4,6); black (0,0)	(704) (705); (706) (707); 0 (708) 1 (709) 2 (710) 3 (711); 4 (712)	0 (background and border) 1; 2 3; 4 5 6 7; 8	(704)	(704)	7900
11	80	192	16 hues, 1 luminance	luminance (0,6)	4	numbers 0–15 give 16 different hues	always black	always black	7900

APPENDIX B: ATASCII CODE CHART

Using Color

Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character / Color Register
0	CTRL-,	♥	Space / 1
1	CTRL-A	♣	! / 1
2	CTRL-B	♦	" / 1
3	CTRL-C	♠	# / 1
4	CTRL-D	♠	$ / 1
5	CTRL-E	♠	% / 1
6	CTRL-F	♠	& / 1
7	CTRL-G	♠	' / 1
8	CTRL-H	♠	(/ 1
9	CTRL-I	♠	) / 1
10	CTRL-J	♠	* / 1
11	CTRL-K	♠	+ / 1
12	CTRL-L	♠	, / 1
13	CTRL-M	♠	- / 1
14	CTRL-N	♠	. / 1
15	CTRL-O	♠	/ / 1
16	CTRL-P	♠	0 / 1
17	CTRL-Q	♠	1 / 1
18	CTRL-R	♠	2 / 1
19	CTRL-S	♠	3 / 1
20	CTRL-T	♠	4 / 1
21	CTRL-U	♠	5 / 1
22	CTRL-V	♠	6 / 1
23	CTRL-W	♠	7 / 1
24	CTRL-X	♠	8 / 1
25	CTRL-Y	♠	9 / 1
26	CTRL-Z	♠	: / 1
27	ESC\ESC	♠	; / 1
28	ESC\CTRL--	♠	< / 1
29	ESC\CTRL-=	♠	= / 1

	Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character	Color Register
134	30	ESC\CTRL-+	←	>	1
135	31	ESC\CTRL-*	→	?	1
33	32	SPACE BAR	Space	Space	0
95	33	SHIFT-1	!	!	0
94	34	SHIFT-2	"	"	0
90	35	SHIFT-3	#	#	0
66	36	SHIFT-4	$	$	0
93	37	SHIFT-5	%	%	0
91	38	SHIFT-6	&	&	0
115	39	SHIFT-7	'	'	0
112	40	SHIFT-9	(	(	0
114	41	SHIFT-0	)	)	0
7	42	*	*	*	0
6	43	+	+	+	0
32	44	,	,	,	0
14	45	-	-	-	0
34	46	.	.	.	0
38	47	/	/	/	0
50	48	0	0	0	0
31	49	1	1	1	0
30	50	2	2	2	0
26	51	3	3	3	0
24	52	4	4	4	0
29	53	5	5	5	0
27	54	6	6	6	0
51	55	7	7	7	0
53	56	8	8	8	0
46	57	9	9	9	0
	58	SHIFT	:	:	0
2	59	;	;	;	0
54	60	<	<	<	0
15	61	=	=	=	0

To get red letters
10 A = Peek(764)
20 ? A 30 G. To 10

	Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character	Color Register
55	62	>	>	>	0
N02	63	SHIFT-\	?	?	0
N17	64	SHIFT-8	@	@	0
N63	65	A	A	A	0
21	66	B	B	B	0
N6	67	C	C	C	0
58	68	D	D	D	0
42	69	E	E	E	0
56	70	F	F	F	0
N61	71	G	G	G	0
57	72	H	H	H	0
13	73	I	I	I	0
11	74	J	J	J	0
5	75	K	K	K	0
ϕ	76	L	L	L	0
37	77	M	M	M	0
35	78	N	N	N	0
46	79	O	O	O	0
N0	80	P	P	P	0
147	81	Q	Q	Q	0
140	82	R	R	R	0
62	83	S	S	S	0
45	84	T	T	T	0
1	85	U	U	U	0
16	86	V	V	V	0
146	87	W	W	W	0
22	88	X	X	X	0
143	89	Y	Y	Y	0
23	90	Z	Z	Z	0
96	91	SHIFT-;)	[	[	0
70	92	SHIFT-, +	\	\	0
96	93	SHIFT-+ •	]	]	0
71	94	SHIFT-*	^	^	0

	Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Mode 1 and 2 Character	Color Register
78	95	SHIFT–	–	–	0
162	96	CTRL-.	◆	@	1
	97	a	a	A	1
	98	b	b	B	1
	99	c	c	C	1
	100	d	d	D	1
	101	e	e	E	1
	102	f	f	F	1
	103	g	g	G	1
	104	h	h	H	1
	105	i	i	I	1
	106	j	j	J	1
	107	k	k	K	1
	108	l	l	L	1
	109	m	m	M	1
	110	n	n	N	1
	111	o	o	O	1
	112	p	p	P	1
	113	q	q	Q	1
	114	r	r	R	1
	115	s	s	S	1
	116	t	t	T	1
	117	u	u	U	1
	118	v	v	V	1
	119	w	w	W	1
	120	x	x	X	1
	121	y	y	Y	1
	122	z	z	Z	1
130	123	CTRL-;	♠	[	1
79	124	SHIFT =	|	\	1
162	125	ESC\CTRL-< or ESC\SHIFT-<	↖	CLEAR SCREEN	
	126	ESC\BACK S	◀	^	1
	127	ESC\TAB	▶	–	1
	128	(↑) CTRL-, ;	♥	Space	3

44 TAB
60 CAPS
39 inverse video

Inverse video

Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character	Color Register
129	(⅄) CTRL-A	A	!	3
130	(⅄) CTRL-B	B	"	3
131	(⅄) CTRL-C	C	#	3
132	(⅄) CTRL-D	D	$	3
133	(⅄) CTRL-E	E	%	3
134	(⅄) CTRL-F	F	&	3
135	(⅄) CTRL-G	G	'	3
136	(⅄) CTRL-H	H	(	3
137	(⅄) CTRL-I	I	)	3
138	(⅄) CTRL-J	J	*	3
139	(⅄) CTRL-K	K	+	3
140	(⅄) CTRL-L	L	,	3
141	(⅄) CTRL-M	M	-	3
142	(⅄) CTRL-N	N	.	3
143	(⅄) CTRL-O	O	/	3
144	(⅄) CTRL-P	P	0	3
145	(⅄) CTRL-Q	Q	1	3
146	(⅄) CTRL-R	R	2	3
147	(⅄) CTRL-S	S	3	3
148	(⅄) CTRL-T	T	4	3
149	(⅄) CTRL-U	U	5	3
150	(⅄) CTRL-V	V	6	3
151	(⅄) CTRL-W	W	7	3
152	(⅄) CTRL-X	X	8	3
153	(⅄) CTRL-Y	Y	9	3
154	(⅄) CTRL-Z	Z	:	3
155	(⅄) RETURN	(EOL) RETURN	BLANK COLUMN	
156	ESC\SHIFT-BACK S		<	3
157	ESC\SHIFT->		=	3
158	ESC\CTRL-TAB		>	3
159	ESC\SHIFT-TAB		?	3
160	(⅄) SPACE BAR	Space	Space	2

1'29

#16
.119

Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character	Color Register
161	(⅄) SHIFT-1	!	!	2
162	(⅄) SHIFT-2	"	"	2
163	(⅄) SHIFT-3	#	#	2
164	(⅄) SHIFT-4	$	$	2
165	(⅄) SHIFT-5	%	%	2
166	(⅄) SHIFT-6	&	&	2
167	(⅄) SHIFT-7	'	'	2
168	(⅄) SHIFT-9	(	(	2
169	(⅄) SHIFT-0	)	)	2
170	(⅄) *	*	*	2
171	(⅄) +	+	+	2
172	(⅄) ,	,	,	2
173	(⅄) -	-	-	2
174	(⅄) .	.	.	2
175	(⅄) /	/	/	2
176	(⅄) 0	0	0	2
177	(⅄) 1	1	1	2
178	(⅄) 2	2	2	2
179	(⅄) 3	3	3	2
180	(⅄) 4	4	4	2
181	(⅄) 5	5	5	2
182	(⅄) 6	6	6	2
183	(⅄) 7	7	7	2
184	(⅄) 8	8	8	2
185	(⅄) 9	9	9	2
186	(⅄) SHIFT-:	:	:	2
187	(⅄) ;	;	;	2
188	(⅄)<	<	<	2
189	(⅄) =	=	=	2
190	(⅄) >	>	>	2
191	(⅄) SHIFT-/	?	?	2
192	(⅄) SHIFT-8	@	@	2
193	(⅄) A	A	A	2

Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character	Color Register
194	(♣) B	B	B	2
195	(♣) C	C	C	2
196	(♣) D	D	D	2
197	(♣) E	E	E	2
198	(♣) F	F	F	2
199	(♣) G	G	G	2
200	(♣) H	H	H	2
201	(♣) I	I	I	2
202	(♣) J	J	J	2
203	(♣) K	K	K	2
204	(♣) L	L	L	2
205	(♣) M	M	M	2
206	(♣) N	N	N	2
207	(♣) O	O	O	2
208	(♣) P	P	P	2
209	(♣) Q	Q	Q	2
210	(♣) R	R	R	2
211	(♣) S	S	S	2
212	(♣) T	T	T	2
213	(♣) U	U	U	2
214	(♣) V	V	V	2
215	(♣) W	W	W	2
216	(♣) X	X	X	2
217	(♣) Y	Y	Y	2
218	(♣) Z	Z	Z	2
219	(♣) SHIFT-,	[	[	2
220	(♣) SHIFT-+	\	\	2
221	(♣) SHIFT-.	]	]	2
222	(♣) SHIFT-*	^	^	2
223	(♣) SHIFT—	—	—	2
224	(♣) CTRL-.	♦	@	3
225	(♣) a	a	A	3
226	(♣) b	b	B	3

Decimal Code	Keystrokes to Produce Character	Mode O ATASCII Character	Modes 1 and 2 Character	Color Register
227	(⅄) c	c	C	3
228	(⅄) d	d	D	3
229	(⅄) e	e	E	3
230	(⅄) f	f	F	3
231	(⅄) g	g	G	3
232	(⅄) h	h	H	3
233	(⅄) i	i	I	3
234	(⅄) j	j	J	3
235	(⅄) k	k	K	3
236	(⅄) l	l	L	3
237	(⅄) m	m	M	3
238	(⅄) n	n	N	3
239	(⅄) o	o	O	3
240	(⅄) p	p	P	3
241	(⅄) q	q	Q	3
242	(⅄) r	r	R	3
243	(⅄) s	s	S	3
244	(⅄) t	t	T	3
245	(⅄) u	u	U	3
246	(⅄) v	v	V	3
247	(⅄) w	w	W	3
248	(⅄) x	x	X	3
249	(⅄) y	y	Y	3
250	(⅄) z	z	Z	3
251	(⅄) CTRL-;	♠	[	3
252	(⅄) SHIFT-=	|	\	3
253	~~ESC\~~CTRL-2	(Buzzer)	]	3
254	(⅄) ~~ESC~~ CTRL-BACK S	(Delete character)	^	3
255	(⅄) ESC\CTRL->	(Insert character)	_	3

156

SE

1 Tones of yellow

2 - screen colors

4 - border

Gray	0	Medium Blue	8	
Light Orange (Gold)	1	Light Blue	9	grayish blue
Orange	2	Turquoise	10	
Red-Orange	3	Green-*Blue*	11	- applegreen
Pink	4	Green	12	
Lavender	5	Yellow-*Green*	13	
Blue-*Purple*	6	Orange-*Green*	14	- golden lighter than #1
Purple-*Blue*	7	Light Orange	15	

deep blue

PLATE 4 Chart of available hues using Atari SETCOLOR command.

16 hues

5338

Alfred Handy Guides

Practical, economical, and concise
Alfred Handy Guides tell you what you need to know quickly and easily—without a lot of reading!

Perfect for today's fast-moving adult on the run, they fit anywhere—in pocket, purse, gadget bag, guitar case. Always where you need them!

The Alfred Handy Guide Series to Computers

How to Buy a Personal Computer
How to Buy a Word Processor
How to Use VisiCalc/SuperCalc
Understanding APL
Understanding Artificial Intelligence
Understanding Atari Graphics
Understanding BASIC
Understanding COBOL
Understanding Data Base Management
Understanding FORTRAN
Understanding LISP
Understanding Pascal

Other Alfred Handy Guide Series

Cooking
Health
Music
Photography

Look for new titles and new series.
For more information:

Alfred Publishing Co., Inc.
P.O. Box 5964
15335 Morrison St.
Sherman Oaks, CA 91413

ISBN 0-88284-224-2